**Εργαστήριο 4**

**05/11/2020**

**ΚΩΔΙΚΑΣ C KAI ASSEMBLY ΣΤΟΝ ATMEL AVR**

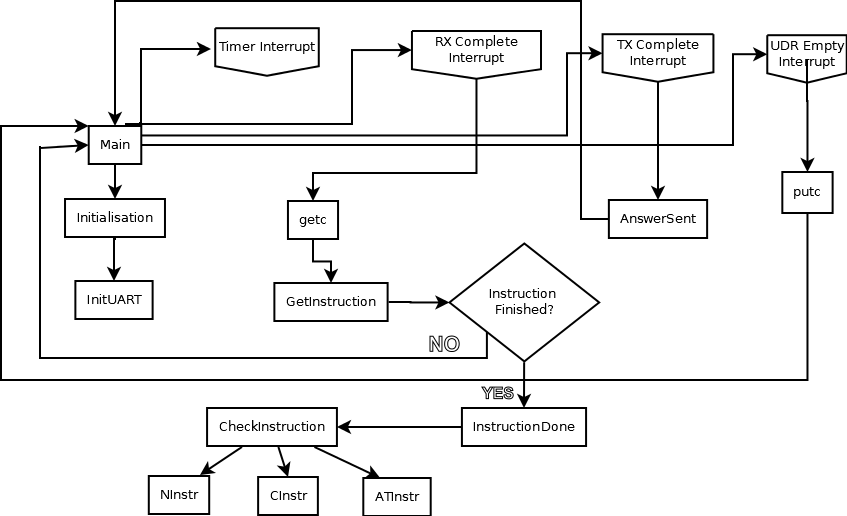
Ντουνέτας Δημήτρης

ΑΜ: 2016030141

# Εισαγωγή

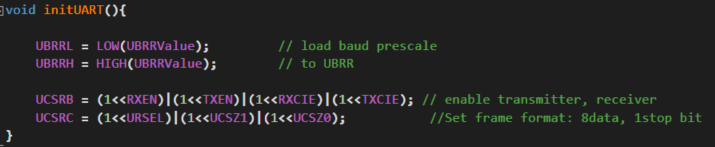
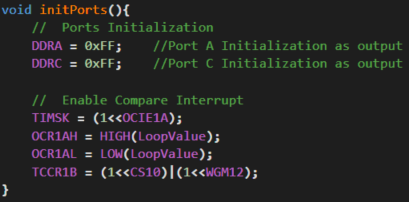
Σκοπός του εργαστηρίου είναι η εξοικείωση με τη γλώσσα C, τον Compiler AVR GCC καθώς και το AVRLIBC πακέτο για προγραμματισμό Μικροελεγκτών AVR. Επιπλέον, η κατανόηση της χρήση C σε συνδυασμό με γλώσσα assembly για πιο Optimized κώδικες. Τέλος, παρατήρηση της assembly εξόδου από τον C Compiler και πως κάνει χρήση των πόρων του συστήματος. Δημιουργία προγράμματος για επικοινωνία με χρήση USART του ATMEGA16, για παραλαβή εντολών και αποστολή απαντήσεων σε συνδυασμό με οδήγηση με πολυπλεξία στον χρόνο μίας οθόνης 7-segment LED για (έως) οκτώ ψηφία.

# Block Diagram Προγράμματος



# Αρχικοποίηση του προγράμματος σε Γλώσσα C

Αρχικά το πρόγραμμα ξεκινάει από την συνάρτηση main που βρίσκεται στο αρχείο main.c . Η main συνάρτηση είναι υπεύθυνη για την αρχικοποίηση του προγράμματος και της μνήμης καθώς και τον ατέρμονα βρόγχο πάνω στον οποίο τρέχει το πρόγραμμα. Για γίνει η αρχικοποίηση η Main καλεί τη συνάρτηση initialization. O Stack Pointer δεν χρειάζεται να αρχικοποιηθεί αφού τον αρχικοποιεί ο C Compiler από μόνος του ώστε να μπορούμε να επιστρέφουμε σωστά από τις ρουτίνες και τις συναρτήσεις που καλούμε.

Η Initialization στην συνέχεια καλεί τις initUΑRT, initRam και initPorts συνάρτήσεις οι οποίες κάνουν τις αρχικοποιήσεις μνήμης , καταχωρητών, θέσεων μνήμης και Ports όπως αναφέρθηκαν στο προηγούμενο εργαστήριο. Συνοπτικά χρησιμοποιούνται 3 σημεία στην μνήμη SRAM στα οποία το ένα με όνομα display\_address είναι το σημείο στο οποίο γράφονται τα BCD στοιχεία που θα δείξει η οθόνη. Το δεύτερο είναι το UARTAnsAddress όπου γράφεται η αυτοματοποιημένη απάντηση που δίνει ο Μικροελεγκτής μας όταν λάβει ένα ολόκληρο Instruction σωστά δηλαδή OK<CR><LF>. Τέλος το τρίτο είναι το InstrAddress όπου αποθηκεύεται το instruction που λαμβάνουμε ώστε να μπορούμε να το διαχειριστούμε κατάλληλα όταν θέλουμε να εκτελέσουμε την εντολή.

# RAM MAP

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0x70 | 0x71 | 0x72 | 0x73 | 0x74 | 0x75 | 0x76 | 0x77 |
| 7segbit7(LSB) | 7segbit6 | 7segbit5 | 7segbit4 | 7segbit3 | 7segbit2 | 7segbit1 | 7segbit0(MSB) |

Η λέξη που βρίσκεται στη θέση 0x70 είναι το δεξιότερο Digit που θα εμφανιστεί στο

7seg-Screen.

|  |  |  |  |
| --- | --- | --- | --- |
| 0x80 | 0x81 | 0x82 | 0x83 |
| UARTAnsByte7(O) | UARTAnsByte6(K) | UARTAnsByte5(CR) | UARTAns4(LF) |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x90 | 0x91 | 0x92 | 0x93 | 0x94 | 0x95 | 0x96 | 0x97 | 0x98 |
| instrByte0 | instrByte1 | instrByte2 | instrByte3 | instrByte4 | instrByte5 | instrByte6 | instrByte7 | instrByte8 |

|  |  |  |
| --- | --- | --- |
| 0x99 | 0x9A | 0x9B |
| instrByte9 | instrByte10 | instrByte11 |

Στις θέσεις ορισμένες ως instrByte εισάγεται διαδοχικά η λέξη της κάθε εντολής που λαμβάνουμε.

|  |  |  |
| --- | --- | --- |
| 0x100 | 0x101 | 0x102 |
| RingCounterAddress | DigitRegAddress | DigitCounterAdress |

# Διαδικασία μετατροπής του προγράμματος από Assembly σε C/AVR.libc assembly mix.

Αρχικά για να τη μεταφορά του προγράμματος από απλή assembly σε μίξη assembly με C γλώσσα χρειάζεται η αλλαγή του επιθέματος του προγράμματος που είναι γραμμένο σε assembly από .asm σε .S . Στη συνέχεια είναι αναγκαία η αλλαγή όλων των Assembly Declarations σε C Declarations.

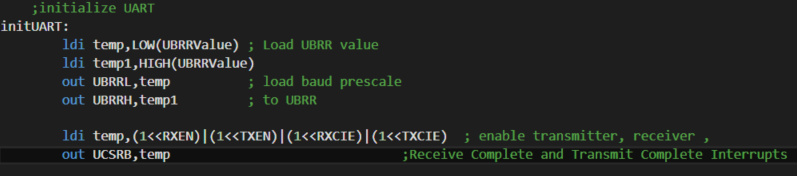
Για την κλήση μιας συνάρτησης Assembly από τον C κώδικα χρειάζεται να δηλωθεί η συνάρτηση ως .global στον κώδικα assembly.

Τέλος σημαντική προσοχή πρέπει να δοθεί στη χρήση καταχωρητών αφού τους χρησιμοποιεί ο C compiler και έτσι δεν μπορούν πλέον να χρησιμοποιηθούν εύκολα ως μεταβλητές. Καλή πρακτική είναι η αποθήκευση των στοιχείων που θέλουμε να χρησιμοποιήσουμε ως μεταβλητές στην μνήμη και η προσπέλαση αυτών από τη μνήμη όποτε χρειάζεται.

Αναλυτικά ο C compiler κάνει push τους καταχωρητές R18 – R31 στην Stack πριν από την κλήση κάθε εντολής assembly οπότε αυτούς τους καταχωρητές μπορούμε να τους χρησιμοποιήσουμε άφοβα στις συναρτήσεις που γράφουμε στην assembly.

# InitUART

Η προσθήκη σε αυτό το εργαστήριο είναι η InitUART όπου φορτώνουμε στον καταχωρητή UBRR την κατάλληλη τιμή που έχουμε υπολογίσει μέσω του Datasheet και σύμφωνα με BAUR Rate 9600 και συχνότητα επεξεργαστή 10Mhz είναι 64. Αυτή η τιμή εισάγεται στον DownCounter του USART ώστε να γίνει το κατάλληλο Prescaling για να λάβουμε το σωστό Baud Rate. Στη συνέχεια, ενεργοποιούμε τον Transmitter και τον Receiver Καθώς και τα Interrupts του βάζοντας 1 στα Bits RXEN, TXEN, RXCIE, TXCIE που βρίσκονται στον καταχωρητή UCSRC.

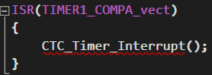


## **Λειτουργία του κύριου προγράμματος και τα Ιnterrupt του USART(UART).**

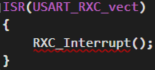
Καθώς το πρόγραμμα έχει ξεκινήσει και βρίσκεται στον ατέρμονα βρόγχο καλούνται τα κατάλληλα interrupts και αυτά χρησιμοποιούμε για να δώσουμε λειτουργικότητα στο πρόγραμμα μας. Αυτή τη στιγμή υπάρχουν 4 Interrupts τα οποία αξιοποιεί το πρόγραμμα για να δείξει στην οθόνη ότι εμείς του δίνουμε ως είσοδο μέσω του UART. Τα Interrupts αυτά είναι:

1. Timer/Counter Compare1 Interrupt
2. USART RX Complete Interrupt
3. USART DATA Registry Empty Interrupt
4. USART TX Complete Interrupt

* To Timer/Counter 1 Compare A Interrupt είναι υπεύθυνο για το σωστό χρονισμό της οθόνης.

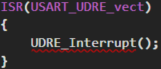


* Καλεί την συνάρτηση CTC\_TIMER\_Interrupt η οποία τρέχει στον Αssembly.S και μας δίνει το κατάλληλο χρονικό περιθώριο για να ανανεώνουμε την 7segment οθόνη για να δώσουμε τα FPS που θέλουμε.
* Το USART RX Complete Interrupt είναι υπεύθυνο για το σωστό διάβασμα όταν σηκωθεί το Flag RXC το οποίο υποδηλώνει ότι έχουμε λάβει κάποιον χαρακτήρα έτοιμο να διαβάσουμε.



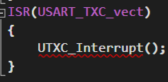
Καλεί την συνάρτηση RXC \_Interrupt η οποία καλεί την getc, που τρέχει στο Assembly.S αρχείο και λαμβάνει ένα χαρακτήρα και τον αποθηκεύει. Θα εξηγηθεί παρακάτω περαιτέρω.

* Το USART DATA Registry Empty Interrupt είναι υπεύθυνο να ενημερώνει όταν το UDR καταχωρητής είναι διαθέσιμος για εγγραφή ώστε να μην απανογράφουμε χωρίς να έχουν αποσταλεί πρώτα τα δεδομένα που θέλουμε.



Καλεί την συνάρτηση UDRE\_Interrupt η οποία καλεί την putc που τρέχει στο Assembly.S αρχείο και στέλνει έναν χαρακτήρα στο UDR ώστε αυτός να αποσταλεί ορθά από τον USART.

* To USART TX Complete Interrupt είναι υπεύθυνο για την ενημέρωση της αποστολής ενός χαρακτήρα. Μας ενημερώνει ότι η αποστολή έχει ολοκληρωθεί και ο χαρακτήρας έχει φτάσει στον προορισμό του. Η διαφορά με το DATA Registry Empty Interrupt είναι ότι περιμένει μέχρι το Byte να φύγει εντελώς από τον Swift Register που αποστέλλει σειριακά και έτσι μας είναι χρήσιμο μόνο σε HALF-Duplex πρωτόκολλα ή όταν θέλουμε να κάνουμε ενέργεια μετά το πέρας της αποστολής. Έτσι το χρησιμοποιούμε για να οριστικοποιήσουμε την αποστολή όλης της απάντησης του Μικροελεγκτή μας.

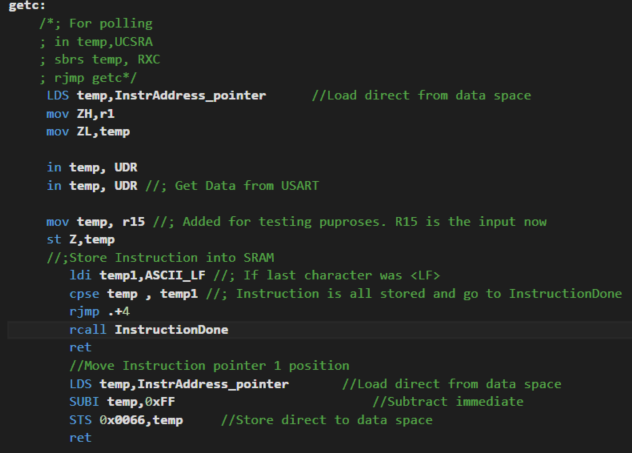


Καλεί την ρουτίνα AnswerSent η οποία τρέχει στο Assembly.S αρχείο και καθαρίζει τη θέση μνήμης που είναι αποθηκευμένη η εντολή ώστε να είναι έτοιμη να δεχτεί την επόμενη εντολή.

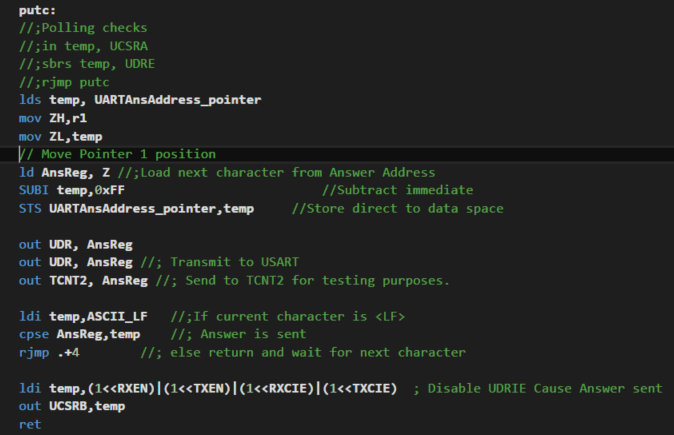
# Παρακάτω γίνεται επεξήγηση των ρουτίνων που τρέχουν στην Assembly από τα προηγούμενα εργαστήρια για λόγους πληρότητας.

# Η Ρουτίνες getc και putc

Η ρουτίνα getc τραβάει ένα Byte που έχει σταλεί μέσω του USART το αποθηκεύει σειριακά στην κατάλληλη θέση μνήμης από τo Reserved σημείο InstrAddress και στη συνέχεια ελέγχει αν έχει ολοκληρωθεί η αποστολή του Instruction σύμφωνα με τους χαρακτήρες <CR> και <LF>.

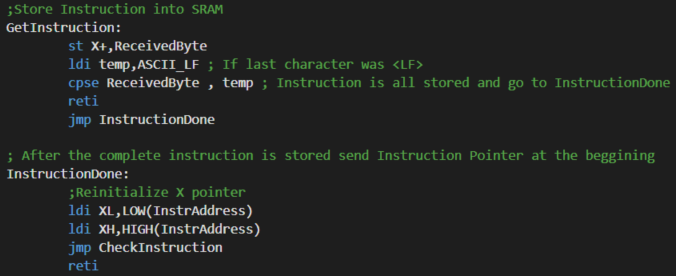


Η ρουτίνα putc αποστέλει ένα χαρακτήρα στον USART. Τραβάει διαδοχικά τους χαρακτήρες από τη θέση μνήμης που είναι γραμμένη η απάντηση μέχρι να τραβήξει τον χαρακτήρα <LF> όπου κλείνει το Interrupt UDRΕ αφού η αποστολή δεν έχει άλλο χαρακτήρα προς μετάδοση.

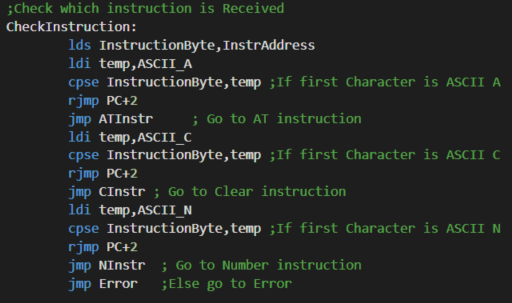


# Διαχείριση εντολών

Για την διαχείριση των εντολών που λαμβάνουμε μέσω του UART υπάρχουν ξεχωριστές συναρτήσεις που ενεργούν αφού λάβουμε σωστά ολόκληρη την εντολή και ουσιαστικά μας έρθει ο χαρακτήρας CR ακολουθούμενος από τον LF. Τότε πηγαίνουμε στην θέση InstructionDone όπου ετοιμάζουμε τους δείκτες για τον έλεγχο της εντολής που μας ήρθε.

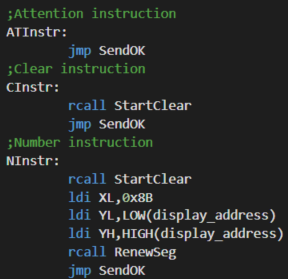


Η ρουτίνα Checkinstruction ελέγχει τί είδους εντολή μας ήρθε από τα 3 δυνατά σενάρια που έχουμε και στέλνει το πρόγραμμα στον κώδικα για τη διαχείριση της.



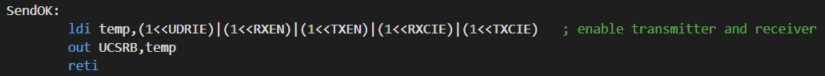
Επειδή θεωρούμε ότι δεν έχουμε λάθος στη μετάδοση και οι εντολές έρχονται πάντα σωστά μπορούμε να απλοποιήσουμε τους ελέγχους που κάνουμε και απλά να ελέγξουμε το πρώτο χαρακτήρα που λάβαμε ο οποίος είναι διαφορετικός για κάθε εντολή.

Έτσι καταλήγουμε σε μια από τις παρακάτω εντολές όπου κάνουμε τις κατάλληλες ενέργειες.



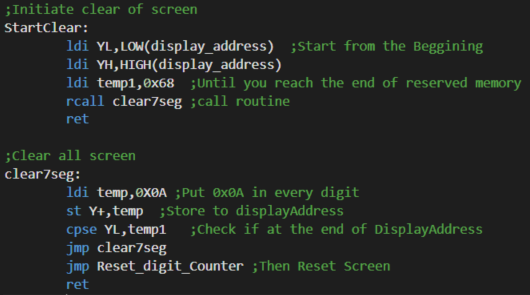
# ΑΤΤΕΝΤΙΟΝ Εντολή

Όταν λάβουμε την εντολή AT<CR><LF> αρκεί να στείλουμε πίσω μια απάντηση ΟΚ. Αυτό εκτελεί η SendOK στην οποία ενεργοποιούμε το Interrupt για να στείλουμε την απάντηση.



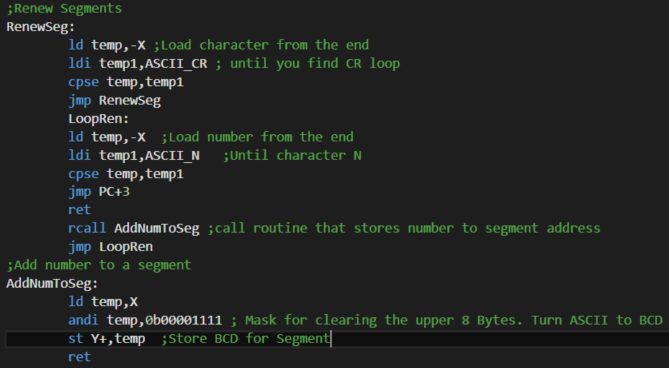
# CLEAR Εντολή

Όταν λάβουμε την εντολή C<CR><LF> αρκεί να καθαρίσουμε την οθόνη ώστε να μη δείχνει τίποτα και να στείλουμε πίσω την απάντηση OK. Έτσι καλούμε την ρουτίνα StartClear και μετά την SendOK.



# NUMBER Εντολή

Όταν λάβουμε την εντολή Ν239…<CR><LF> αρκεί να καθαρίσουμε την οθόνη ώστε να μη δείχνει τίποτα ύστερα να βάλουμε τα σωστά ψηφία στην οθόνη και να στείλουμε πίσω την απάντηση OK. Καθαρίζουμε πηγαίνουμε στη συνάρτηση RenewSeg και στέλνουμε την απάντηση όπως προηγούμενα.



Aυτή η ρουτίνα διαβάζει την αποθηκευμένη εντολή από το τέλος και αφού βρει που ξεκινάνε τα νούμερα που έχουμε ως ορίσματα τα αποθηκεύει 1-1 κατάλληλα στις θέσεις μνήμης ώστε να τα δείχνει σωστά η οθόνη 7seg. Δηλαδή το δεξιότερο νούμερο το αποθηκεύει στο δεξιότερο 7seg και τα υπόλοιπα αντίστοιχα.

# Τελικό Αποτέλεσμα

Το πρόγραμμα τρέχει συνεχόμενα χωρίς να σταματάει αφού ως βασικό μέρος έχει έναν ατέρμονα βρόγχο. Διαβάζει εντολές και ενεργεί κατάλληλα καθώς επίσης απαντάει και OK αφού διαβάσει μια σωστή εντολή.

# Παρατήρηση

Επειδή το Atmel Studio δεν έχει κατάλληλο περιβάλλον για αποσφαλμάτωση USART επικοινωνίας έχουν γίνει κάποιες αλλαγές στο πρόγραμμα ώστε να είναι δυνατή η προσομοίωση του με STIMFILE και να logάρονται κατάλληλα οι απαντήσεις. Η εισαγωγή των bytes που λαμβάνονται γίνεται με τον Καταχωρητή 15 και έτσι τα διαβάζουμε ως είσοδο στο πρόγραμμά μας. Επίσης οι απαντήσεις δίνονται στον Καταχωρητή TCNT2 αυθαίρετα. Θα μπορούσαμε να χρησιμοποιήσουμε οποιουσδήποτε καταχωρητές δεν χρησιμοποιούμε απλά το Stim File δεν μπορεί να διαβάσει τον UDR ούτε τα PINS RX και TX.

Στην διαδικασία αποσφαλμάτωσης παρατηρήθηκε επίσης ότι κατά τη διάρκεια αποστολής συμβόλου που χρησιμοποιούμε το Flag UDRE για την ενεργοποίηση του Interrupt αυτό το Flag όπως και το RXC χρειάζεται να το προσπελάσουμε και να το γράψουμε 2 φορές και να ανταποκριθεί το USART καθώς και ότι αφού σταλούν δεδομένα χρειάζεται περίπου 180.000 κύκλους για να σηκωθεί το επόμενο Flag και να καλεστεί το interrupt. Αυτό σύμφωνα με τους υπολογισμούς που λαμβάνουμε υπόψιν το Baud Rate θα έπρεπε να συμβαίνει κάθε 10.000 κύκλους χοντρικά. Πιθανόν να είναι σφάλμα που οφείλεται στο Atmel Studio είτε στην τρόπο που θεωρεί το Stim File ρολόι επειδή αυτή η διαφορά μιας τάξης μεγέθους θα μπορούσε να δικαιολογηθεί αν το ρολόι είχε τιμή κοντά στο 1 MHz.